Data Structures and Algorithms [JAVA]
(KK)

[V1 → V7] ⇒ Java Basics (covered in cwh, refer previous Java notes)

V.8 Arrays and Arraylist

* Collection of Data types is known as Array (in Simple terms)

Eg.-

```
int roll1 = 23;
int roll2 = 55;
int roll3 = 18;
  .
  .
  .
int roll---.---;
```

Dynamic Memory Allocation (compile time / run time)

```
int[] rollnos = new int[5];
```

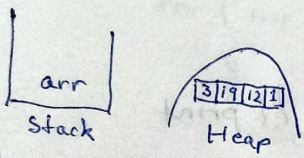
Array name (ref. variable) — Array size — new creates object.

(or)

```
int[] rollnos = {23, 12, 45, 19---};
```

This primitive defines the datatype of the array.

One array can store only one datatype.



* Array objects are in heap
* Heap objects are not continuous (It is continuous in case of C/C++)
* Dynamic Memory Allocation.

Hence: In Java, Array may not be continuous (Depends on JVM)

Index of Array:

0	1	2	3	4	5	6
8	3	19	12	7	28	33

```
print(arr[0]) ⇒ 8
arr[2] ⇒ 19
arr[3] = 99 ⇒ Updates the values from 12 to 99
```

In case of string,

```
String[] arr = new String[4];
```

(or)

```
String[] arr = {"hi", "hello", "bye"};
```

(**) Empty array (int) returns 0.
Empty array (String) returns null.

Input using for loop :

```
int[] arr = new int[5];
for (int i = 0; i < arr.length; i++)
{
    arr[i] = in.nextInt();
}
for (int i = 0; i < arr.length; i++)  // print
{
    sout(arr[i] + " ");
}
```

(or)

```
// print
{
    sout(Arrays.toString(arr));
}
```

Multi-Dimensional (2D) Array :→

```
int[][] arr = new int[3][3];
```

(or)

↑ No. of columns is not mandatory.

```
int[][] arr = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};
```

(**) It is stored as array of arrays internally.

```
int [][] arr = new int [3][2];
sout (arr.length);      // no. of rows.
for (int row = 0; row < arr.length; row++)      // input
    {
      for (int col = 0; col < arr[row].length; col++)
        {
          arr [row][col] = in.nextInt ();
        }
    }

for (int row = 0; row < arr.length; row++)      // output
    {
      for (int col = 0; col < arr [row].length; col++)
        {
          sout (arr.[row] [col] + " ");
        }
      sout ();
    }

                (or)

for (int row = 0; row < arr.length; row++)      // output
  {
    sout (Arrays.toString (arr[row]));
  }
```

Array list :→

(*) Syntax → `ArrayList <Integer> list = new ArrayList <> (10);`

```
list.add (71);
list.add (32);
  :
and so on.   sout (list);
```

(*) The array size is fixed internally in Arraylist, but it creates a new arraylist, which is double the size and copies the older elements in to the new arraylist. The old arraylist gets deleted and it keeps on happening.

So, it appears like it has no fixed size

(*) Array Questions :

1) Swapping Values in an Array →

```
{
    int[] arr = {1, 3, 23, 9, 18};
        swap(arr, 1, 3);
        sout(Arrays.toString(arr));
}

static void swap(int[] arr, int index1, int index
    {
        int temp = arr[index1];
        arr[index1] = arr[index2];
        arr[index2] = temp;
    }
```

Output : [1, 9, 23, 3, 18]

2) Max value in an Array →

```
{   int[] arr = {1, 3, 23, 9, 18};
    sout(max(arr));
}
```

```
static int max (int [] arr)
    {
      int maxVal=arr [0];
     for (int i = 1 ; i<arr.length ; i++)
        {   if (arr[i] > maxVal)
              {  maxVal = arr [i];
              }
        }return maxVal;
    }
```

3) Reverse an Array →

```
{
   int [] arr = { 1, 3, 23, 9, 18};
    reverse (arr);
    sout (Arrays. to String (arr));
}
static void reverse (int [] arr)
{     int start = 0;
      int end = arr. length - 1;
     while (start < end)
        {
          swap (arr, start, end);
          start ++;
          end --;
        }
    }
Static void swap ( int [] arr , int index1, int index2
    {   int temp = arr [index 1];
        arr [index 1] = arr [index 2];
        arr [index 2] = temp;
    }
```